ladanaque@utec.edu.pe
CS5372: Network and Computer Security
October 15, 2025

L3 - Software Security

Instrucciones:

- Esta actividad solo debe ejecutarse en un entorno local aislado (máquina virtual o contenedor en una red privada). No desplegar las aplicaciones vulnerables en internet ni en infraestructuras de producción. El propósito es educativo y defensivo.
- **Desarrollar de forma individual**
- Debe adjuntar todos los archivos correspondientes, sino no tendrá calificación.

Objetivos:

- Entender los conceptos de command injection y SQL injection.
- Construir y probar vulnerabilidades en un entorno controlado.
- Implementar y verificar contramedidas (validación, prepared statements, uso seguro de subprocess, least privilege).
- Añadir mecanismos básicos de detección (logging, patrones / reglas simples) y evaluar su eficacia.

Requerimientos:

- Docker y docker-compose instalados en la máquina del estudiante (o VM local).
- Navegador web.
- Python 3.10+ (solo para lectura / testing local si no se usa Docker).
- Opcional: herramienta de inspección HTTP (curl, Postman) y herramienta IDS/WAF simple (ModSecurity si se quiere ampliar).

Procedimiento

Question 1. Crear un proyecto con dos servicios en Docker: una app web vulnerable (Flask + SQLite) y una versión parcheada para comparar, crear dos carpetas: vulnapp/ y fixedapp/, cada una con su Dockerfile y código. Para la vulnapp seguir estas recomendaciones:

- /ping: usa subprocess.getoutput() con interpolación de cadena shell=True implícito, permite inyectar operadores de shell (;, &&).
- /user: construye la consulta SQL mediante concatenación, permitiendo ' OR '1'='1 y otras inyecciones.

Para la fixdeapp aplicar las siguientes contramedidas:

- Para command injection: validación del input y uso de subprocess.run() con lista de argumentos (sin shell).
- Para SQL injection: ? placeholders (parameterized queries).

Question 2. Crear la estructura de carpetas y archivos descritos (vulnapp/ y fixedapp/), desde la carpeta raíz ejecutar:

```
docker-compose build
docker-compose up
```

Question 3. Acceder mediante:

- App vulnerable: http://localhost:5000/user?username=alice.
- App parcheada: http://localhost:5001/user?username=alice.

Question 4. Pruebas controladas:

- SQL Injection (vulnapp): Payload para ver todos los usuarios:

```
/user?username=' OR '1'='1
```

Este payload observa una query devuelta y rows. ¿Cómo se comporta la app parcheada con el mismo payload?

- Command Injection (vulnapp): Payload para ver todos los usuarios:

```
/ping?host=127.0.0.1; ls o
/ping?host=127.0.0.1 && id
```

Verificar que en la app parcheada la validación y el uso de lista de argumentos debe impedir la ejecución de ls o id.

Question 5. Detección de errores mediante logs

- Login detallado: Registra IP origen, endpoint, parámetros, user-agent, timestamp, resultado de verificación o error. En Python: usar logging para escribir a fichero/STDOUT con rotación (RotatingFileHandler).
- Reglas: Detectar caracteres típicos de inyección en parámetros (', ", ;,–,/*,*/,&&,|).
- Regla IDS básica (pseudocódigo): si un mismo IP hace solicitudes con patrones sospechosos en n intentos se debe bloquear temporalmente y enviar alerta.
- Puede usar este regex ejemplo (para parámetros que no deberían contener esos tokens):

```
import re
suspicious_re = re.compile(r"[\';\-\-\/\*]|&&|\||`")
if suspicious_re.search(param_value):
    alert("Posible inyeccion", details)
```

Question 6. Detección de errores SQL

- Registrar mensajes de error SQL que contienen stacktraces o fragmentos de consulta. Un alto número puede indicar escaneo de inyecciones.

Question 7. Técnicas de prevención Investigar la pertinencia de usar las siguientes técnicas de prevención:

- SQL: usar siempre prepared statements / parameterized queries, o un ORM confiable; evitar concatenación de SQL.
- Command: nunca ejecutar comandos construidos por concatenación; usar APIs que acepten lista de argumentos (subprocess.run([...])) y evitar shell=True.
- Validación y whitelisting: validar entrada por tipo, longitud y caracteres permitidos; preferir whitelisting sobre blacklisting.
- Least privilege: limitar permisos de la cuenta que ejecuta la app y de la BD.
- Cifrado y rotación: proteger backups y credenciales; rotar claves.
- Mitigaciones adicionales: WAF, rate limiting, CAPTCHA en endpoints sensibles, MFA para operaciones críticas.

Entregas

- Sesión 15/10: App vulnerable y app parcheada, asegurarse de que tenga una determinada cantidad de accesos.
- 21/10: Dockers y accesos creados para la vulnapp y la fixedapp.
- 22/10: Pruebas y redacción del informe con resultados.